IOB2-2

Digital Product Development

TU Delft

Module 4
Data

By Alessandro Bozzon

PART 1

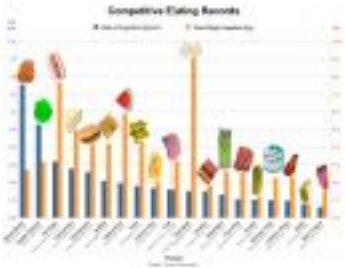
Learning Objectives

- Understand the reasons for using a Database and the importance of Database Management Systems
- Describe the main characteristics of a DBMS
- Understand the importance of data abstraction and modelling for data management systems
- Describe and design SQL programs for the retrieval of data from tables
- Prototype database applications using open-source database systems (e.g., SQLite)

Why Databases?

MANAGEMENT OF INFORMATION

- Information is handled and recorded according to various techniques:
  - Informal ideas
  - Natural language (written or spoken)
  - Drawings, Diagrams
  - Numbers
  - Codes
- As activities become **systematised**, appropriate forms of organisation and **codification** for information have been devised
- E.g. Information about people



MANAGEMENT OF INFORMATION

In most computer-based systems (including software-based products) information is **represented** by means of data

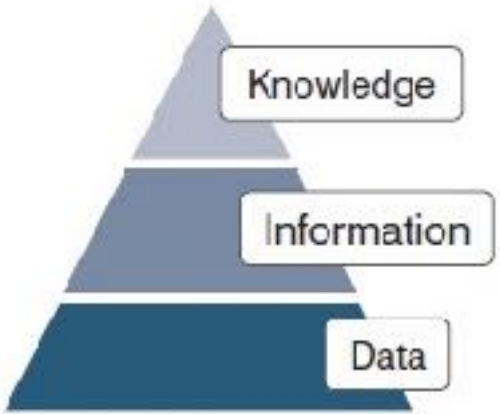
"*Alessandro*" and 2 are a string and a number – two pieces of **data**

If they are provided as a reply to a request

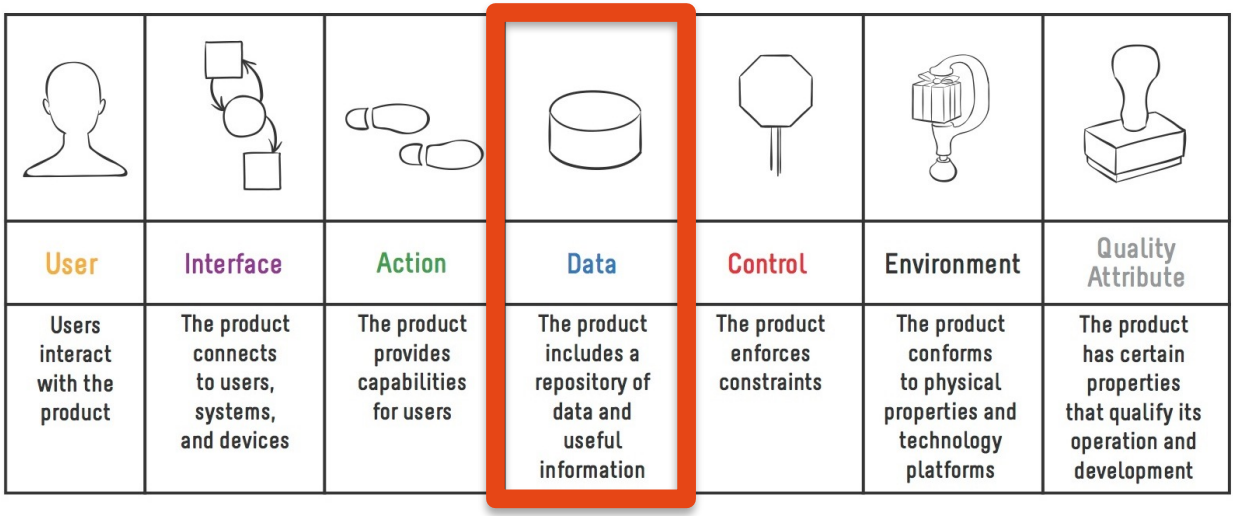
- "2nd year course lectured by a teacher named Alessandro"

then we get information out of them

When we combine this information with other details from the CourseBase, we gain "*understanding*" of the didactic offer of the IDE Faculty for bachelor students



7 Product Dimensions



User	Interface	Action	Data	Control	Environment	Quality Attribute
Users interact with the product	The product connects to users, systems, and devices	The product provides capabilities for users	The product includes a repository of data and useful information	The product enforces constraints	The product conforms to physical properties and technology platforms	The product has certain properties that qualify its operation and development

- The Data dimension describes the data and useful information the product stores and uses.
- This includes data needed by actions and data sent and received via product interfaces.

WHY IS DATA SO IMPORTANT?

Data is at the very core of every information systems

- No organisation can operate without a good strategy for data management and access
- It is a core asset, and it must be managed and protected

Data has a longer life-cycle than the processes and organisations that manage them

- e.g. bank data management systems did not change in decades (or centuries)
- But the processes that manage them change every year

DEFINITIONS OF DATABASE

- **Generic Definition**

> A collection of interrelated data, used to represent information of interest to an information system

  - Represents some aspects of the real world: the **Universe of Discourse (UoD)**
    - The problem setting under consideration
    - It consider the point of view of an actor (or a set of actors) in the system
    - A database is logically coherent in such setting, and it has meaning in it
  - Is shared between different software applications and different users

- **More Technical Definitions**

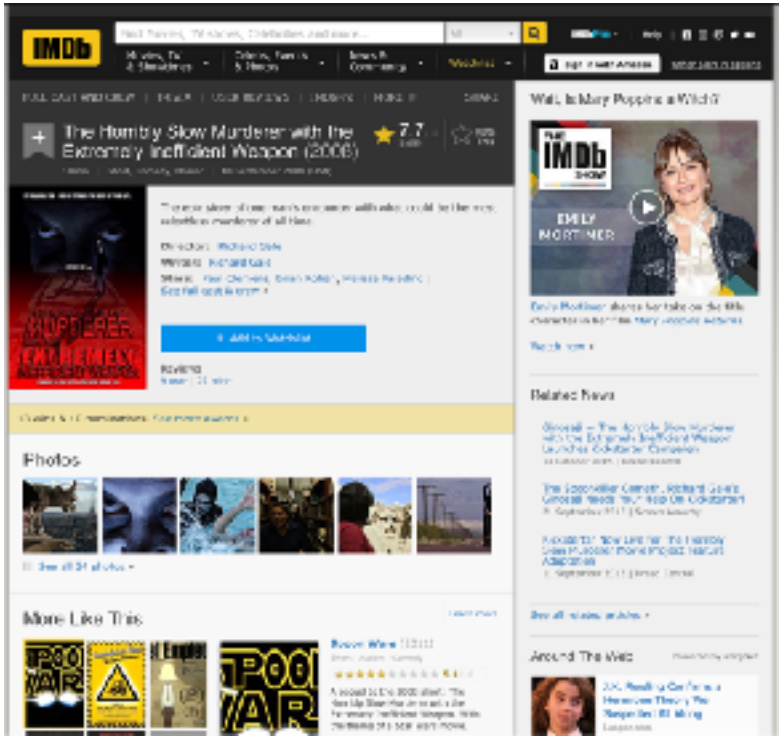
> A collection of files that store the **data**

Database Management Systems (DBMS)

DEALING WITH DATA

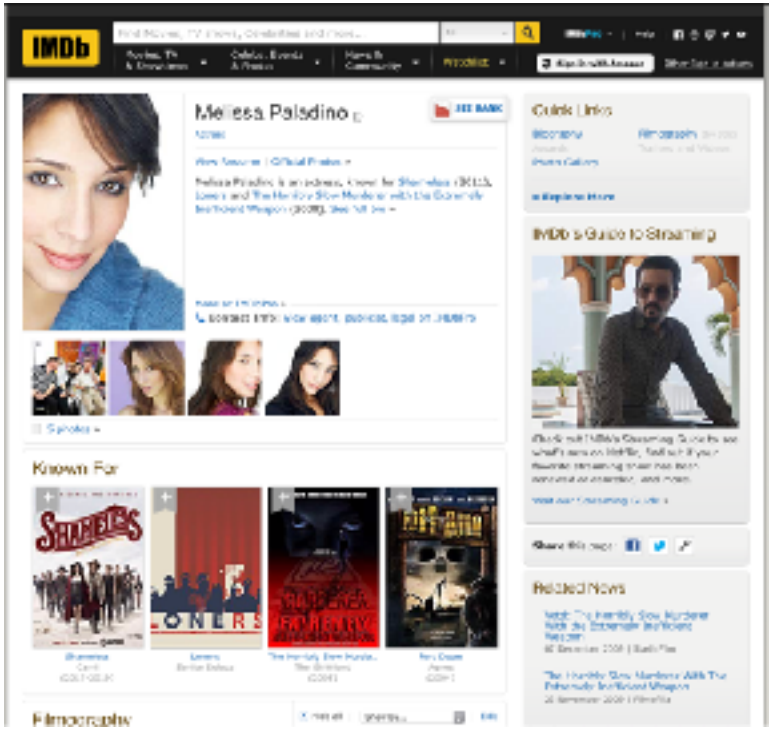
- How to manage large and persistent sets of data?

https://www.imdb.com/title/tt1301160/



- 5M Titles
- A movie page contains information about actors, their roles in the movie, etc.

https://www.imdb.com/name/nm1574516/



- 9M Names
- An actor's page contains biographic information, their roles in movie, etc.

DEALING WITH DATA

- How to manage large and persistent sets of data?
  - File systems
  - Store the data in files

- Developers define and implement the files needed for a specific software
  - A description of the organisation of the files (often just a stream of bytes)
  - Application logic to access / query / update files content

movies.csv

movieid, movietitle, year, roles

4200972,Providence,1991,||Clinton Oie||Poetry Host||Keanu Reeves||Eric||Yvonne de la Vega||Herself||Tracii Show||Trish

4477062,The Matrix,1999,||Julian Arahanga||Apoc||David Aston||Rhineheart||Jeremy Ball||Businessman||Michael Butcher||Cop Who Captures Neo||Marcus Chong||Tank||Steve Dodd||Blind Man||Matt Doran||Mouse||Mike Duncan||Twin||Nash Edgerton||Resistance Member||Laurence Fishburne||Morpheus||Paul Goddard||Agent Brown||Marc Aden Gray||Choi||Nigel Harbach||Parking Cop||Harry Lawrence||Old Man||Bernard Ledger||Big Cop||David O'Connor||FedEx Man||Joe Pantoliano||Cypher||Anthony Ray Parker||Dozer||Chris Pattinson||Cop||Luke Quinton||Security Guard||Keanu Reeves||Neo||Robert Simper||Cop||Robert Taylor||Agent Jones||Hugo Weaving||Agent Smith||Adryn White||Potential||Rowan Witt||Spoon Boy||Lawrence Woodward||Guard||Bill Young||Lieutenant||Tamara Brown||Potential||Gloria Foster||Oracle||Deni Gordon||Priestess||Fiona Johnson||Woman in Red||Belinda McClory||Switch||Rana Morrison||Shaylae - Woman in Office||Carrie-Anne Moss||Trinity||Ada Nicodemou||Dujour||Janaya Pender||Potential||Natalie Tjen||Potential||Eleanor Witt||Potential

actors.csv

actorid, name, gender,year, movies

2213918,Robert Simper,m,||The Matrix||Cop

3797028,Miranda Richardson,f,||The Evening Star||Patsy Carpenter||Muppets Most Wanted||Berliner at Window

1411109,Enrico Lo Verso,m,||Hannibal||Gnocco

2897965,Tanya Champoux,f,||The Day the Earth Stood Still||Isabel

4022321,Kathy Graves Toon,f,||Horton Hears a Who!||Additional Voice

3004466,Debi Derryberry,f,||Horton Hears a Who!||Who Mom||Horton Hears a Who!||Additional Voices||Jimmy Neutron: Runaway Rocketboy!||Jimmy Neutron

591926,Johnny Depp,m,||Close Up||Himself||London Fields||Chick Purchase||Rock and a Hard Place: Another Night at the Agora||Himself||Pearl Jam Twenty||Himself||The Rum Diary||Kemp

1179714,Israel Juarbe,m,||The Net||Thief

2854981,Carol Burnett,f,||Horton Hears a Who!||Kangaroo

2017355,Bertrand Roberson Jr.,m,||The Day the Earth Stood Still||Soldier

1974351,Blair Redford,m,||The Day the Earth Stood Still||Army Fighter Pilot #1

567375,Robert De Niro,m,||Close Up||Himself||Les cent et une nuits de Simon Cinma||Le mari de la star-fantasme en croisire||The Audition||Robert De Niro||I sogni nel mirino||Himself||Lennon or McCartney||Himself||Stardust||Captain Shakespeare

3715153,Janaya Pender,f,||The Matrix||Potential

1564754,Jake McLaughlin,m,||The Day the Earth Stood Still||Soldier

Retrieve the name and role of actresses that played in “The Matrix”

query.js

```
var fs = require("fs");

var actors = require('fs').readFileSync('actors.csv').toString().match(/^.+$/gm);
var movies = require('fs').readFileSync('movies.csv').toString().match(/^.+$/gm);

for (i = 0; i < movies.length; i++){
  title = movies[i].split(',')[1]
  year = movies[i].split(',')[2]
  if (title == 'The Matrix'){
    console.log('Title:', title)
    console.log('--- Women in Cast ---')
    var cast = movies[i].split(',')[3]
    var actors_in_movie = cast.split('||')
    for (j = 1; j < actors_in_movie.length; j=j+2){
      for (k = 0; k < actors.length; k++){
        name = actors[k].split(',')[1]
        gender = actors[k].split(',')[2]
        if(name == actors_in_movie[j] && gender == 'f'){
          console.log(name, 'as', actors_in_movie[j+1])
        }
      }
    }
  }
};
```

```
Title: The Matrix
--- Women in Cast ---
Tamara Brown as Potential
Gloria Foster as Oracle
Deni Gordon as Priestess
Fiona Johnson as Woman in Red
Belinda McClory as Switch
Rana Morrison as Shaylae - Woman in Office
Carrie-Anne Moss as Trinity
Ada Nicodemou as Dujour
Janaya Pender as Potential
Natalie Tjen as Potential
Eleanor Witt as Potential
```

WHAT IF . . .

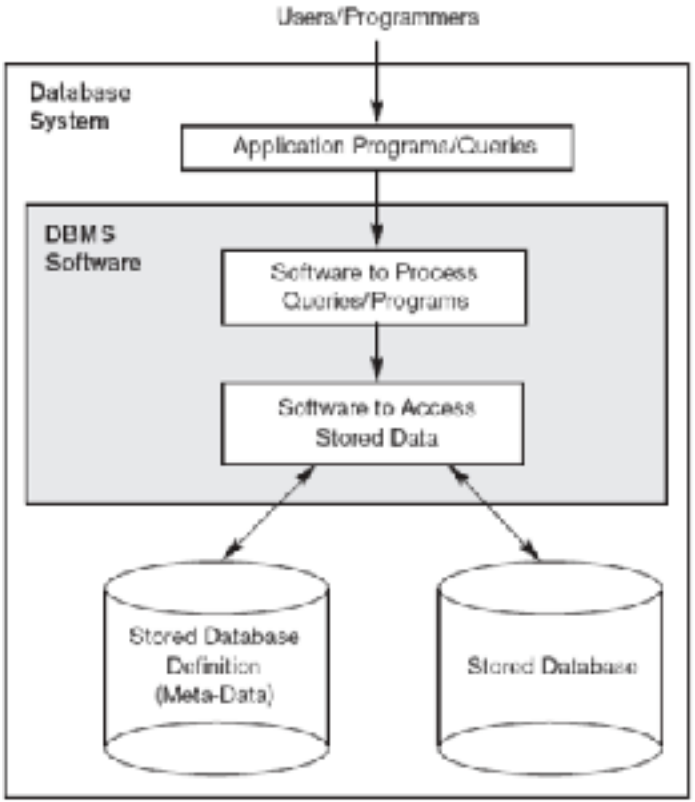
- Several applications make use of the same data
- The system crashes
- The dataset grows (let's say, up to 100 Tb)
- Many users want to access to the data, possibly concurrently
- Information needs are not pre-defined
- ....

We need a more **efficient** and **effective** solution:
A **Database Management System**

DATABASE MANAGEMENT SYSTEM

A software system able to manage **collections of data** that are

- **Large**: bigger, much bigger that the main memory available
- **Shared**: used by various applications and users
- **Persistent**: with a lifespan that is not limited to single executions of the programs that use them



MAIN FUNCTIONS OF A DBMS

- **Queries**
  - To *retrieve* data that match certain selection criteria expressed in the **query**
  - Queries *do not change* the state of the database
- **Transactions**
  - To *insert*, *delete*, and *update* data in the database.
  - Transactions *change* the state of the database
- **Security**
  - *Authentication*, i.e. verification of the identity of a client application
  - *Authorisation*, i.e. the enforcement of access and execution rules for queries and transactions
  - More later

C reate

R ead

U pdate

D elete

TYPICAL FEATURES OF DBMSs

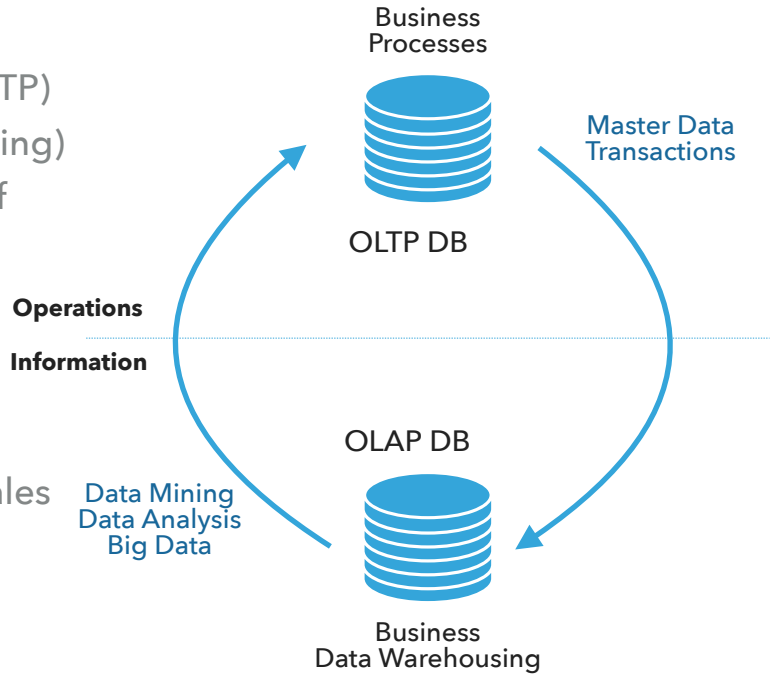
- **Data Integrity** and **Evolution**
  - Durability, Integrity, Correctness, Evolvability
- **Performance**
  - Scalability, Elasticity, Latency, Throughput, Partition Tolerance
- **Security** and **Privacy**
  - Security, Confidentiality, Non-Repudiation

ADDITIONAL ADVANTAGES OF DBMSs

- Reduced application development time
- Economies of scale
- Efficient query processing
- Several (built-in or external) user interfaces
- Self-describing nature
    - DBMSs might contain complete definition of structure (Meta-data) and rules of validity

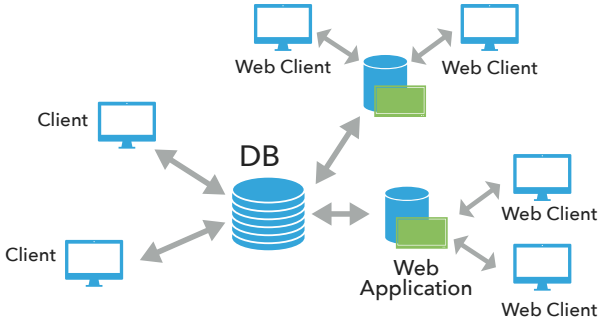
CLASSIFICATION OF DBMSs / TYPE OF USAGE

- **Operational** Databases: OnLine Transaction processing (OLTP)
  - Management of dynamic data in real-time (e.g. banking)
  - Emphasis on transaction efficiency and on support of daily operations
  - Main concern: concurrency
  - Users: personnel, end users
- **Analytical** Databases: OnLine Analytical Processing (OLAP)
  - Interactive analysis of multi-dimensional data (e.g. sales reports)
  - Emphasis on data integration and aggregation
  - Main concerns: storage and query execution time
  - Users: managers, executive, data scientists

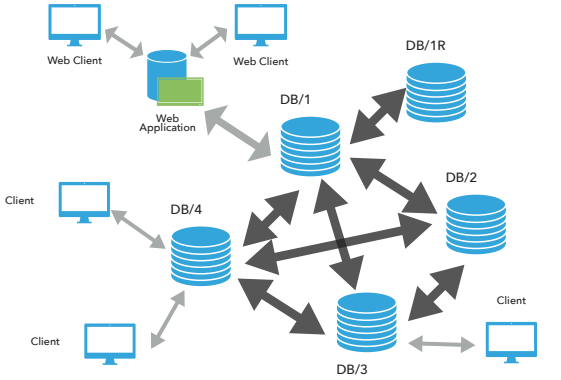


CLASSIFICATION OF DBMSs / DISTRIBUTION

- **Centralised**
  - Database is located, stored, and maintained in a single computer
  - Advantages: minimal redundancy, better security and preservation
  - Disadvantages: single point of failure, scalability / elasticity



- **Distributed**
  - Database and the DBMS software are distributed from various sites that are connected by a computer network
  - Advantages: availability, scalability / elasticity, redundancy
  - Disadvantages: complexity, security, data integrity



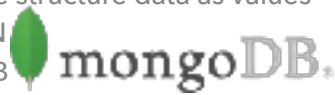
CLASSIFICATION OF DBMSs / DATA MODEL

Relational

- Based on the relational model (Codd, 1970)
- Data organised in homogeneous set of tuples (rows) forming relations (tables)
- SQL as generic data definition, manipulation and query language for relational data
- Ensure atomicity, consistency, isolation, and durability (ACID paradigm)
- Examples: PostgreSQL, MySQL, SQL Server, SQLite, Oracle, MariaDB, etc.

Non - Relational

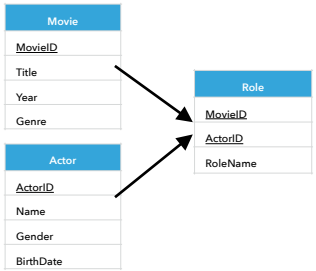
- **Key-Value**
  - Records only containing a key and a value. The key uniquely identify the record, the value is an arbitrary chunk of data
  - Examples: Amazon Dynamo, Redis
- **Document-oriented**
  - Similar to key/value, but require structure data as values using formats like XML or JSON
  - Examples: MongoDB, CouchDB



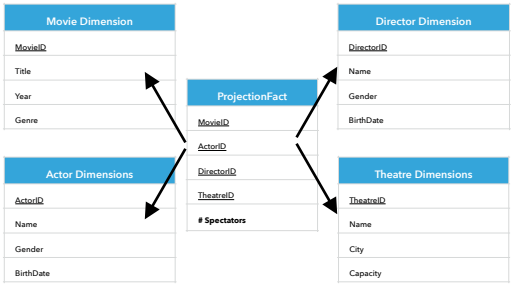
- **Wide-Column**
  - Store data by Columns-Families, rather than by row
  - Examples: Google BigTable, DynamoDB, Apache Cassandra
- **Graph**
  - Data organised in graphs. Nodes describe main data entities, edges describe relationships.
  - Example: Neo4J

CLASSIFICATION OF DBMSs / DATA MODEL

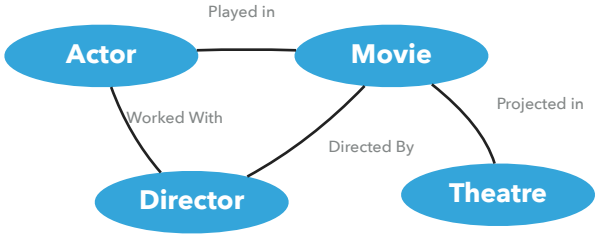
Relational



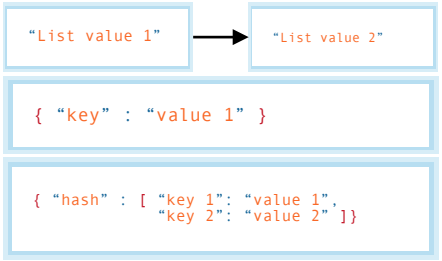
Dimensional



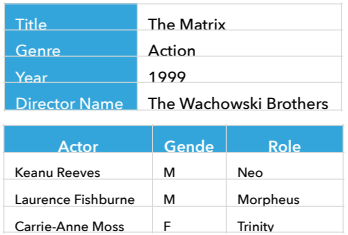
Graph



Key-Value



Document



Title	The Matrix
Genre	Action
Year	1999
Director Name	The Wachowski Brothers

Actor	Gende	Role
Keanu Reeves	M	Neo
Laurence Fishburne	M	Morpheus
Carrie-Anne Moss	F	Trinity

Wide-Column

Key	MovieID	Movie Title	Actor Name	Role	Director	Theatre
1	1	The Matrix	Keanu Reeves	Neo		
2	1	The Matrix	Carrie-Anne Moss	Trinity		
3	1	The Matrix			The Wachowski Brothers	
4	1	The Matrix				Pathe' Delft

WHEN NOT TO USE A DBMS

- More desirable to use regular files for:
  - Simple, well-defined database applications not expected to change at all
  - Stringent, real-time requirements that may not be met because of DBMS overhead
  - Embedded systems with limited storage capacity
  - No multiple-user access to data

It is just someone else's C program

In the beginning we may be impressed by its speed

But later we discover that it can be frustratingly slow

We can do any particular task faster outside the DBMS

IOB2-2

Digital Product Development

TU Delft

Module 4
Data

By Alessandro Bozzon

PART 2

Interacting with Databases

DBMS LANGUAGE CLASSES

- Data Definition Language (DDL)
  - Defines the logical and physical schema
  - Often used also for access authorisation specification

SQL | Cypher | MongoDB QL

- Data Manipulation Language (DML)
  - Allows retrieval, insertion, deletion, modification of database instances

SQL | Cypher | MongoDB QL

- Storage Definition Language (SDL)
  - Specifies the internal schema
- View Definition Language (VDL)
  - Specifies user views/mapping to conceptual schema
  - Typically the same as DDL

SQL | MongoDB QL

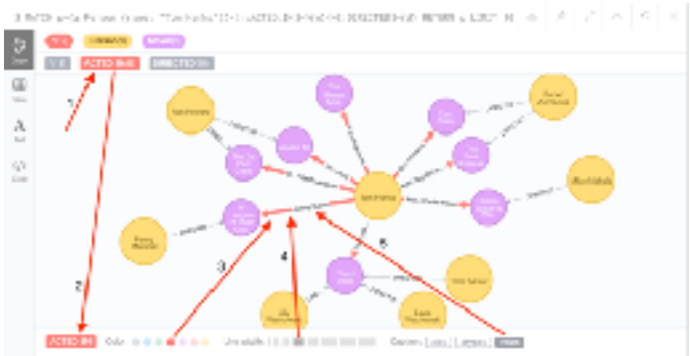
DATABASE LANGUAGES

- Various forms (a contribution to effectiveness)
  - Interactive textual and declarative language
    - **SQL** (Structured Query Language - Relational DBMSs)
    - CYPHER (Neo4J)
    - MongoDB QL (MongoDB)
  - Interactive commands embedded in a host language (Java, C++, Python, Javascript)
- By means of non-textual **user-friendly** interfaces
  - Graphical user interfaces (e.g. PGAdmin, Neo4J Browser)
  - Natural language interfaces
  - Interfaces for the DBA

Neo4J Query

```
MATCH (T:TITLE{
            TITLE_NAME: 'THE MATRIX',
            PRODUCTION_YEAR: '1999'}
      )-[R:HAS_CAST]->(C:CASTINFO)<-[S:PARTICIPATES_IN]-(P:PERSON)

RETURN DISTINCT P.NAME
```



Neo4J Browser UI

THE SQL LANGUAGE

THE SQL LANGUAGE

- The name is an acronym for **S**tructured **Q**uery **L**anguage
- Far richer than a query language: a **DML**, a **DDL**/ **VDL**
- SQL is a **set-based** language
  - operators works on relations (tables)
  - results are always relations (tables)
- SQL is declarative
  - It describes **what** to do with data, not **how** to do it

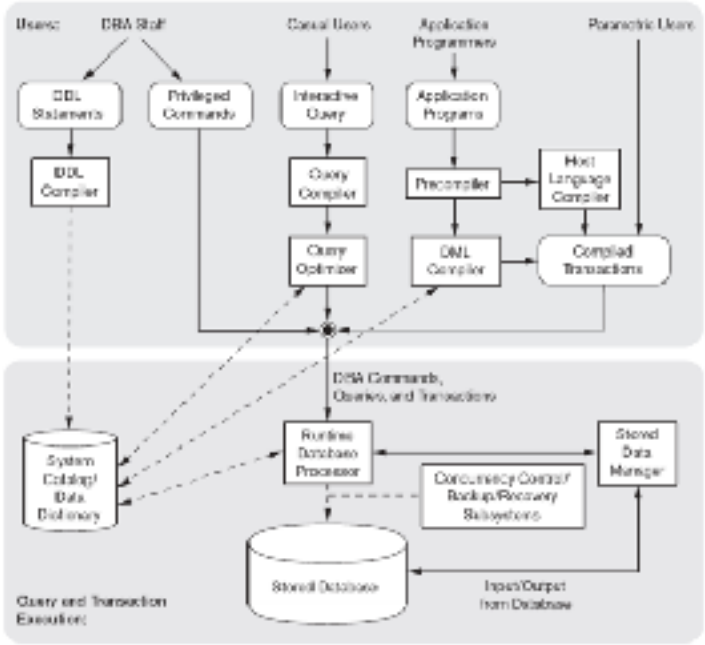


Figure 2.3
Component modules of a DBMS and their interactions.

SQL – STRUCTURED QUERY LANGUAGE

- A text-based declarative language for **relational** databases used as:
  - DDL and VDL by DBMS designer and DBA
  - DML by users

```
SELECT DISTINCT P.NAME
FROM PERSON P JOIN CAST_INFO K ON (P.ID = K.PERSON_ID)
              JOIN TITLE T ON (K.MOVIE_ID = T.ID)
WHERE T.TITLE = 'THE MATRIX' AND T.PRODUCTION_YEAR = 1999;
```

Query: find the name of all the actors that played a part in the movie "The Matrix" produced in 1999

SQL IS INTERGALACTIC DATA SPEAK

- Successful, mainstream, and general purpose 4GL (fourth generation programming language) – perhaps the only one

A brief discussion of this new class of users is in order here. There are some users whose interaction with a computer is so infrequent or unstructured that the user is unwilling to learn a query language. For these users, natural language or menu selection (3,4) seem to be the most viable alternatives. However, there is also a large class of users who, while they are not computer specialists, would be willing to learn to interact with a computer in a reasonably high-level, non-procedural query language. Examples of such users are accountants, engineers, architects, and urban planners. It is for this class of users that SEQUEL is intended. For this reason, SEQUEL emphasizes simple data structures and operations.

Chamberkin, Boyce. http://www.joakimdalby.dk/HTM/sequel.pdf

- Many standards out there:
  - ANSI SQL, SQL92 (a.k.a. SQL2), SQL99 (a.k.a. SQL3), ...
  - Vendors support various subsets (or supersets!)

Credits: Dave Roth

EXAMPLE DATABASES

EXAMPLE DB1: EMPLOYEES

Employee					
FirstName	**Surname**	**Dept**	**Office**	**Salary**	**City**
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

Department		
DeptName	**Address**	**City**
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Joné

EXAMPLE DB2: PRODUCTS

Supplier

CodeS	NameS	Shareholders	Office
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Supply

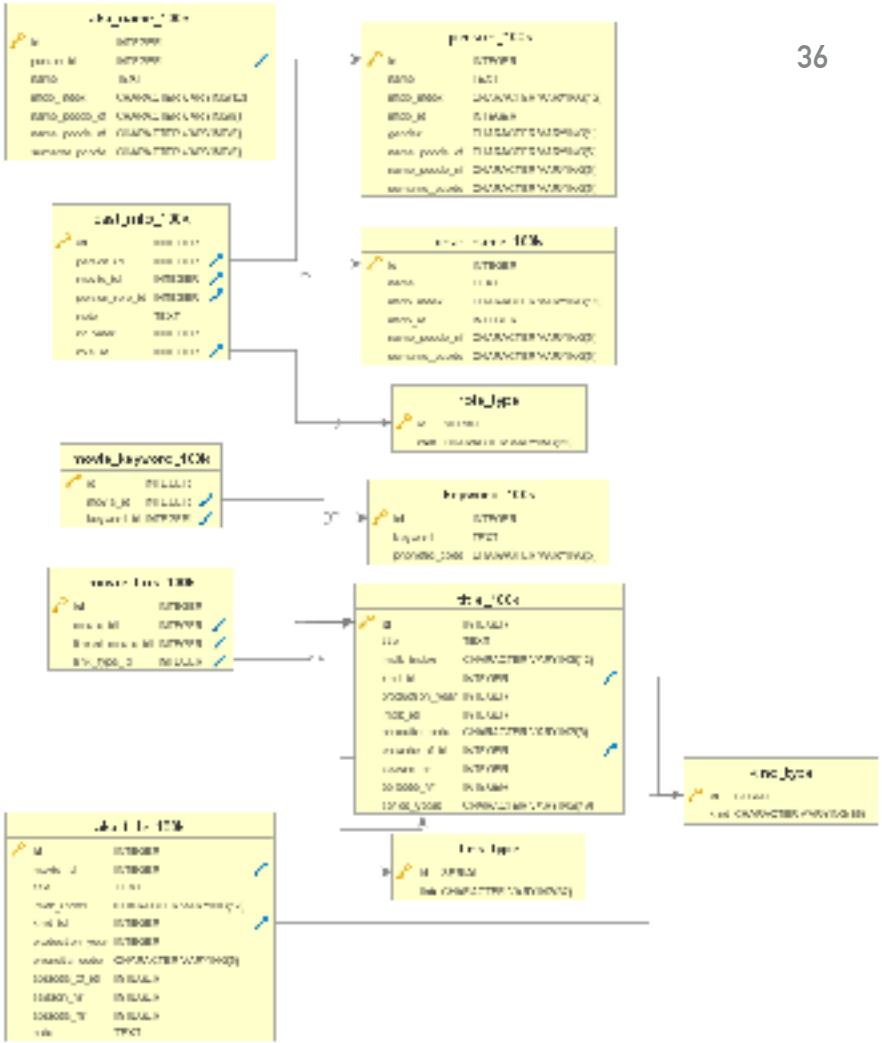
CodeS	CodeP	Amount
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

EXAMPLE DB3: IMDB

- A subset of the schema and data from the IMDB.com website
  - Actors (person_100k), Movies (title_100k), and Actors in Movies (cast_info_100k)
  - Plus aliases, keywords, movie genres, etc.
- Get it (with import instructions) here
  - https://docs.google.com/document/d/1jj3cMAnk6Rc0mHkkOAIYDzYLjKisCuyj4-3KF9l-_8o
- The instructions are for PostgresQL. Reach out to the teaching team if you are interested



QUERYING

SQL AS A QUERY LANGUAGE

- SQL expresses queries in declarative way
  - queries specify the properties of the result, not the way to obtain it
- Queries are translated by the query optimiser into the procedural language internal to the DBMS
- The programmer should focus on readability, not on efficiency

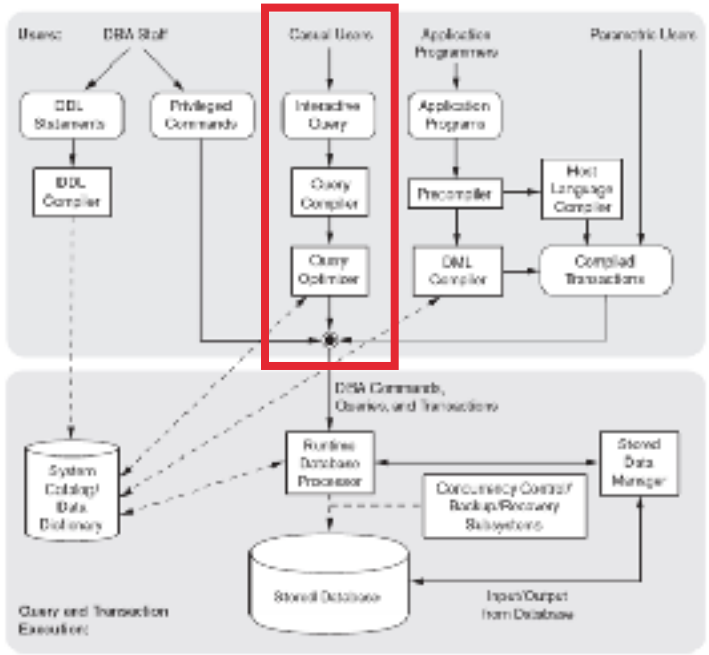


Figure 2.3
Component modules of a DBMS and their interactions.

SQL QUERIES

- Expressed by the SELECT statement

```
SELECT TargetList
FROM Table
[ WHERE Conditions ] [ ORDER BY OrderingAttributesList ]
[ GROUP BY GroupingAttributesList ] [ HAVING AggregateConditions ]
```

- The three parts of the query are usually called:
  - **Target list** or `SELECT` clause
  - `FROM` clause
  - `WHERE` clause
- The query:
  - considers the cartesian product of the tables in the `FROM` clause
  - considers only the rows that **satisfy** (evaluate to `TRUE`) the condition in the `WHERE` clause
  - for each row evaluates the attribute expressions in the `TargetList`, and returns them
  - More on `GROUP BY` and `HAVING` later

SELECT CLAUSE

SELECT QUERY FROM /1

- Find the *code* of **all** products in the DB

```
SELECT CodeP
FROM Products
```

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

CodeP
P1
P2
P3
P4
P5
P6

SELECT QUERY FROM /2

- Find the *code* and *number of shareholders* of suppliers ***located in*** "Den Haag"

```
SELECT CodeS, Shareholders
FROM Supplier
WHERE Office = "Den Haag"
```

Supplier

CodeS	NameS	Shareholders	Office
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

CodeS	Shareholders
S2	1
S3	3

Only the tuples evaluating the logical expression in the `WHERE` clause to `TRUE` are selected

* IN THE TARGET LIST

- Find all the information relating to employees **named** “Brown”

```
SELECT *
FROM Employee
WHERE Surname = “Brown”
```

Employee					
FirstName	**Surname**	**Dept**	**Office**	**Salary**	**City**
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	Brown	Planning	14	80	London

ATTRIBUTE EXPRESSIONS WITH AS/1

- The keyword AS allows the definition of an **alias**. Used in attribute expressions, it defines a new **temporary** column per the calculated expression
- Find the monthly salary of the employees **named** "White"

```
SELECT Salary / 12 AS MonthlySalary
FROM Employee
WHERE Surname = "White"
```

Employee

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

MonthlySalary
3.00

ATTRIBUTE EXPRESSIONS WITH AS/2

- Find the salaries of employees *named* "Brown", and alias it as "Remuneration"

```
SELECT Salary AS Remuneration
FROM Employee
WHERE Surname = "Brown"
```

Employee

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

Remuneration
45
80

DUPLICATES

- In relational algebra and calculus the results of queries **do not contain duplicates** (set semantics)
- In SQL, result sets **may have identical rows** (bag semantics)
- Duplicates **rows** can be removed using the keyword `DISTINCT`
  - This applies also with rows having multiple columns

```
SELECT City
FROM Department
```

Department

DeptName	Address	City
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Joné

City
London
Toulouse
Brighton
London
San Joné

```
SELECT DISTINCT City
FROM Department
```

Department

DeptName	Address	City
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Joné

City
London
Toulouse
Brighton
San Joné

DISTINCT KEYWORD

- Find the code of the products **supplied <u>at least</u> by one supplier**

```
SELECT DISTINCT CodeP
FROM Supply
```

Supply

CodeS	CodeP	Amount
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

CodeP
P1
P2
P3
P4
P5
P6

WHERE CLAUSE

WHERE CLAUSE

- **Selection conditions** apply to **each single tuple** resulting from the evaluation of the `FROM` clause
- Defined as a **boolean expression of simple predicates**
- Simple predicates
  - comparison between attributes and/or constant values
  - set membership
  - textual matching
  - `NULL` values

PREDICATE CONJUNCTION /1

- Find the *first names* and *surnames* of the employees **who work in office number** `20` **of the** `"Administration"` **department**

```
SELECT FirstName, Surname
FROM Employee
WHERE Office = "20" AND Dept = "Administration"
```

Employee

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

FirstName	Surname
Gus	Green

PREDICATE CONJUNCTION /2

- Find the *first names* and *surnames* of the employees **who work in the** “Administration” **department and in the** “Production” **department**

```
SELECT FirstName, Surname
FROM Employee
WHERE Dept = “Administration” AND Dept = “Production”
```

- ???

Employee					
FirstName	**Surname**	**Dept**	**Office**	**Salary**	**City**
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

- No Results!

PREDICATE DISJUNCTION

- Find the *first names* and *surnames* of the employees **who work in <u>either</u> the "Administration" <u>or</u> the "Production" department**

```
SELECT FirstName, Surname
FROM Employee
WHERE Dept = "Administration" OR Dept = "Production"
```

Employee

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

FirstName	Surname
Mary	Brown
Charles	White
Gus	Green
Pauline	Bradshaw
Alice	Jackson

COMPLEX LOGICAL EXPRESSIONS

- Find the first names of the employees **named** "Brown" **who work in the** "Administration" **department or the** "Production" **department**

```
SELECT FirstName
FROM Employee
WHERE Surname = "Brown" AND (Dept = "Administration" OR Dept = "Production")
```

Employee

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

FirstName
Mary

PostgreSQL Logical Operators: https://www.postgresql.org/docs/current/functions-logical.html

OPERATOR LIKE /1

- Matching string patterns
- The character “_” is a matching term for any single character, which must be found in the specified position
- The character “%” is a matching term for any sequence of *zero* or more characters

OPERATOR LIKE /2

- Find the *code* and the *name* of the products **having name starting with the letter** "S"

```
SELECT CodeP, NameP
FROM Products
WHERE NameP LIKE "S%"
```

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

CodeP	NameP
P1	Sweater
P3	Shirt
P4	Shirt
P5	Skirt

OPERATOR LIKE /3

- Find the employees with **surnames that have "r" as the second letter and that end in "n"**

```
SELECT *
FROM Employee
WHERE Surname LIKE "_r%n"
```

Employee

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Gus	Green	Administration	20	40	Oxford
Charles	Brown	Planning	14	80	London

OPERATOR LIKE /4

- Find Suppliers **having the Office address containing the string** "Den Haag"

```
WHERE Address LIKE "%Den Haag%"
```

- Find Suppliers where **the supplier code ends in** 2

```
WHERE CodeS LIKE "_2"
```

- Find Products that are in storehouses **having names that do not contain an** "e" **as second character**

```
WHERE Storehouse NOT LIKE "_e%"
```

PostgreSQL LIKE Operator Reference: https://www.postgresql.org/docs/current/functions-matching.html#FUNCTIONS-LIKE
Other PostgreSQL Pattern Matching Operators: https://www.postgresql.org/docs/current/functions-matching.html

WHICH OF THE FOLLOWING QUERIES RETURN THE SAME RESULT SET?

1

```
SELECT *
FROM char_name_100k
WHERE name = 'A'
```

2

```
SELECT *
FROM char_name_100k
WHERE name LIKE 'A'
```

3

```
SELECT *
FROM char_name_100k
WHERE name LIKE 'A_'
```

4

```
SELECT *
FROM char_name_100k
WHERE name LIKE 'A%'
```

5

```
SELECT *
FROM char_name_100k
WHERE name LIKE 'A%_'
```

A) Only **1)** and **2)**

B) Only **3)** and **4)**

C) **1)** and **2)** , **4)** and **5)**

D) **All**

DEALING WITH NULL VALUES

- NULL values may mean that:
  - a value is **unknown** (exists but it is not known)
  - a value is **not available** (exists but it is purposely withheld)
  - a value is **not applicable** (undefined for this tuple)
- Each individual NULL value is considered to be **different from every other** NULL value
- When a NULL is involved in a comparison operation, the results is considered to be UNKNOWN
- SQL uses a **three-valued logic**
  - TRUE, FALSE, and UNKNOWN
  - All logical operators evaluate to TRUE, FALSE, or UNKNOWN
  - In PostgreSQL, these are implemented as `true`, `false`, and NULL
  - Most of this is common to different SQL database servers, although some servers may return any nonzero

COMPARISONS INVOLVING NULL AND THREE-VALUED LOGIC

Table 5.1 Logical Connectives in Three-Valued Logic

(a)

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

(b)

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(c)

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

THE IS NULL OPERATOR

- AttributeName IS [NOT] NULL
- Find the *code* and the *name* of products **having no specified** Size

```
SELECT CodeP, NameP
FROM Products
WHERE Size IS NULL
```

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	NULL	Den Haag
P6	Coat	Red	42	Amsterdam

CodeP	NameP
P5	Skirt

THE IS NULL OPERATOR

- Find the *code* and the *name* of products **having size greater than** 44, **or that <u>might</u> have size greater than** 44

```
SELECT CodeP, NameP
FROM Products
WHERE Size > 44 OR Size IS NULL
```

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	NULL	Den Haag
P6	Coat	Red	42	Amsterdam

CodeP	NameP
P2	Jeans
P3	Shirt
P5	Skirt

ORDERING OF RESULTS

ORDERING

- The ORDER BY clause, at the end of the query, orders the rows of the results
- Last operator applied by the database in the query execution plan
- Syntax:

```
ORDER BY OrderingAttribute [ asc | desc  ]
          {, OrderingAttribute [ asc | desc  ]}
```

- The implicit ordering is ASC: ascending

ORDER BY /1

- Find the *code*, *name* and the *size* of all the products, **in descending order of size**

```
SELECT CodeP, NameP, Size
FROM Products
ORDER BY Size DESC
```

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

CodeP	NameP	Size
P2	Jeans	48
P3	Shirt	48
P4	Shirt	44
P6	Coat	42
P1	Sweater	40
P5	Skirt	40

ORDER BY /2

- Find all the information about products, **in ascending order of** *name* **and descending order of** *size*

```
SELECT *
FROM Products
ORDER BY NameP, Size DESC
```

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

CodeP	NameP	Color	Size	Storehouse
P6	Coat	Red	42	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P1	Sweater	Red	40	Amsterdam

ORDER BY /3

- Find the code and the *american size* of all the products, **in ascending order of size**

```
SELECT CodeP, Size - 14 AS AmericanSize
FROM Products
ORDER BY AmericanSize
```

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

CodeP	AmericanSize
P1	26
P5	26
P6	28
P4	30
P2	34
P3	34

JOIN

QUERYING MULTIPLE TABLES

- All possible tuple combinations
- What if we want to retrieve:
  - the name of all the **suppliers of product** "P2"

Supplier

CodeS	NameS	Shareholders	Office
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Supply

CodeS	CodeP	Amount
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

CROSS PRODUCT /1

- All possible tuple combinations
- Find the *name* of **all** the suppliers **of** product "P2"

```
SELECT NameS
FROM Supplier, Supply
```

Supplier				Supply		
CodeS	NameS	Shareholders	Office	CodeS	CodeP	Amount
S1	John	2	Amsterdam	S1	P1	300
S1	John	2	Amsterdam	S1	P2	200
S1	John	2	Amsterdam	S1	P3	400
S1	John	2	Amsterdam	S1	P4	200
S1	John	2	Amsterdam	S1	P5	100
S1	John	2	Amsterdam	S1	P6	100
S1	John	2	Amsterdam	S2	P1	300
...	...	...	...	...	...	...
S2	Victor	1	Den Haag	S1	P1	300
...	...	...	...	...	...	...
S2	Victor	1	Den Haag	S2	P1	300
...	...	...	...	...	...	...
S3	Anna	3	Den Haag	S1	P1	300
...	...	...	...	...	...	...
S3	Anna	3	Den Haag	S3	P2	200
...	...	...	...	...	...	...

CROSS PRODUCT /2

- Find the *name* of **all** the suppliers **of** product "P2"

Supplier				Supply		
CodeS	NameS	Shareholders	Office	CodeS	CodeP	Amount
S1	John	2	Amsterdam	S1	P1	300
S1	John	2	Amsterdam	S1	P2	200
S1	John	2	Amsterdam	S1	P3	400
S1	John	2	Amsterdam	S1	P4	200
S1	John	2	Amsterdam	S1	P5	100
S1	John	2	Amsterdam	S1	P6	100
S1	John	2	Amsterdam	S2	P1	300
...	...	...	...	...	...	...
S2	Victor	1	Den Haag	S1	P1	300
...	...	...	...	...	...	...
S2	Victor	1	Den Haag	S2	P1	300
...	...	...	...	...	...	...
S3	Anna	3	Den Haag	S1	P1	300
...	...	...	...	...	...	...
S3	Anna	3	Den Haag	S3	P2	200
...	...	...	...	...	...	...

What is the problem with this result set?

SIMPLE JOIN /1

```
SELECT NameS
FROM Supplier, Supply
WHERE Supplier.CodeS = Supply.CodeS
```

=

Supplier				Supply		
CodeS	NameS	Shareholders	Office	CodeS	CodeP	Amount
S1	John	2	Amsterdam	S1	P1	300
S1	John	2	Amsterdam	S1	P2	200
S1	John	2	Amsterdam	S1	P3	400
S1	John	2	Amsterdam	S1	P4	200
S1	John	2	Amsterdam	S1	P5	100
S1	John	2	Amsterdam	S1	P6	100
S1	John	2	Amsterdam	S2	P1	300
...	...	...	...	...	...	...
S2	Victor	1	Den Haag	S1	P1	300
...	...	...	...	...	...	...
S2	Victor	1	Den Haag	S2	P1	300
...	...	...	...	...	...	...
S3	Anna	3	Den Haag	S1	P1	300
...	...	...	...	...	...	...
S3	Anna	3	Den Haag	S3	P2	200
...	...	...	...	...	...	...

SIMPLE JOIN /2

- `Supplier.CodeS = Supply.CodeS` is a **JOIN CONDITION**

Supplier				Supply		
CodeS	NameS	Shareholders	Office	CodeS	CodeP	Amount
S1	John	2	Amsterdam	S1	P1	300
S1	John	2	Amsterdam	S1	P2	200
S1	John	2	Amsterdam	S1	P3	400
S1	John	2	Amsterdam	S1	P4	200
S1	John	2	Amsterdam	S1	P5	100
S1	John	2	Amsterdam	S1	P6	100
S2	Victor	1	Den Haag	S2	P1	300
S2	Victor	1	Den Haag	S2	P2	400
S3	Anna	3	Den Haag	S3	P2	200
S4	Angela	2	Amsterdam	S4	P3	200
S4	Angela	2	Amsterdam	S4	P4	300
S4	Angela	2	Amsterdam	S4	P5	400

OUR ORIGINAL QUERY

- Find the *name* of **all** the suppliers **of** product "P2"

```
SELECT NameS
FROM Supplier, Supply
WHERE Supplier.CodeS = Supply.CodeS AND CodeP = "P2"
```

Supplier				Supply		
CodeS	NameS	Shareholders	Office	CodeS	CodeP	Amount
S1	John	2	Amsterdam	S1	P1	300
S1	John	2	Amsterdam	S1	P2	200
S1	John	2	Amsterdam	S1	P3	400
S1	John	2	Amsterdam	S1	P4	200
S1	John	2	Amsterdam	S1	P5	100
S1	John	2	Amsterdam	S1	P6	100
S2	Victor	1	Den Haag	S2	P1	300
S2	Victor	1	Den Haag	S2	P2	400
S3	Anna	3	Den Haag	S3	P2	200
S4	Angela	2	Amsterdam	S4	P3	200
S4	Angela	2	Amsterdam	S4	P4	300
S4	Angela	2	Amsterdam	S4	P5	400

NameS
John
Victor
Anna

ANOTHER QUERY

- Find the *name* of supplier of **at least one red product**

```
SELECT DISTINCT NameS
FROM Supplier, Supply, Products
WHERE Supplier.CodeS = Supply.CodeS AND Supply.CodeP = Product.CodePCodeP
      AND Color = “Red"
```

If there are *N* tables in the `FROM` clause, at least *N – 1* `JOIN` conditions in the `WHERE` clause

Supplier

CodeS	NameS	Shareholders	Office
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Supply

CodeS	CodeP	Amount
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

NameS
John
Victor

USING AS KEYWORD FOR TABLES

- Find the *code* **pairs** of suppliers **having their office in the same city**
- All possible tuple combinations

```
SELECT S1.CodeS, S2.CodeS
FROM Supplier AS S1, Supplier AS S2
WHERE S1.Office = S2.Office
```

Supplier AS S1			
CodeS	**NameS**	**Shareholders**	**Office**
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Supplier AS S2			
CodeS	**NameS**	**Shareholders**	**Office**
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

RESULT /1

- Find the *code* **pairs** of suppliers **having their office in the same city**

Supplier AS S1			
CodeS	**NameS**	**Shareholders**	**Office**
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Supplier AS S2			
CodeS	**NameS**	**Shareholders**	**Office**
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

S1.CodeS	S2.CodeS
S1	S1
S1	S4
S2	S2
S2	S3
S3	S2
S3	S3
S4	S1
S4	S4
S5	S5

- **Pairs** of identical values
- **Permutations** of the same pair of values

RESULT /2

- Find the *code* **pairs** of suppliers **having their office in the same city**

```
SELECT S1.CodeS, S2.CodeS
FROM Supplier AS S1, Supplier AS S2
WHERE S1.Office = S2.Office AND S1.CodeS <> S2.CodeS
```

- Remove pairs with the same code value

S1.CodeS	S2.CodeS
~~S1~~	~~S1~~
S1	S4
~~S2~~	~~S2~~
S2	S3
S3	S2
~~S3~~	~~S3~~
S4	S1
~~S4~~	~~S4~~
~~S5~~	~~S5~~

RESULT/3

- Find the *code* **pairs** of suppliers **having their office in the same city**

```
SELECT S1.CodeS, S2.CodeS
FROM Supplier AS S1, Supplier AS S2
WHERE S1.Office = S2.Office AND S1.CodeS < S2.CodeS
```

- Let's keep only the right ones

S1.CodeS	S2.CodeS
~~S1~~	~~S1~~
S1	S4
~~S2~~	~~S2~~
S2	S3
~~S3~~	~~S2~~
~~S3~~	~~S3~~
~~S4~~	~~S1~~
~~S4~~	~~S4~~
~~S5~~	~~S5~~

JOINS IN SQL92

http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt

- SQL-2 introduced an alternative syntax for the representation of JOINs, representing them explicitly in the from clause:

```
SELECT TargetList
FROM Table [[AS] Alias]
   { [JoinType] JOIN Table [[AS] Alias] [ON BooleanExpression || USING JoinColumns]}
[ WHERE Conditions ]
```

- JoinType can be any of `INNER`, `RIGHT [OUTER]`, `LEFT [OUTER]` or `FULL [OUTER]`, permitting the representation of outer joins
- The keyword `NATURAL` may precede `JoinType`

JOINS IN SQL92

- `NATURAL JOIN` on two relations `R` and `S`
  - No join condition specified
  - Implicit `EQUI JOIN` condition for each pair of attribute with same name from `R` and `S`
- `INNER JOIN`
  - **Default** type of join in a joined table (equivalent to `JOIN`)
  - Must specify `JOIN` attributes
  - Tuple is included in the results only if a matching tuple exists in the other relation
- `LEFT OUTER JOIN`
  - Every tuple in **left** table must appear in result
  - If no matching tuple: values for attributes in the right table set to `NULL`
- `RIGHT OUTER JOIN`
  - Every tuple in **right** table must appear in result
  - If no matching tuple: values for attributes in the left table set to `NULL`
- `FULL OUTER JOIN`
  - If no matching tuple: values for attributes in the left and/or right tables set to `NULL`

INNER JOIN

- Find the name of supplier of **at least one red product**

```
SELECT DISTINCT NameS
FROM Products JOIN Supply USING (CodeP)
              JOIN Supplier USING (CodeS)
WHERE Color = “Red"
```

Supplier

CodeS	NameS	Shareholders	Office
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

Supply

CodeS	CodeP	Amount
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

Products

CodeP	NameP	Color	Size	Storehouse
P1	Sweater	Red	40	Amsterdam
P2	Jeans	Green	48	Den Haag
P3	Shirt	Blu	48	Rotterdam
P4	Shirt	Blu	44	Amsterdam
P5	Skirt	Blu	40	Den Haag
P6	Coat	Red	42	Amsterdam

NameS
John
Victor

- Same results as in Slide 58

LEFT OUTER JOIN

- Find the *code* and *name* of Supplier, and the *code* of the supplied Products, showing also suppliers of no products

```
SELECT Supply.CodeS, Supplier.NameS, Supply.CodeP
FROM Supplier LEFT OUTER JOIN Supply
              ON Supplier.CodeS = Supply.CodeS
```

CodeS	NameS	CodeP
S1	John	P1
S1	John	P2
S1	John	P3
S1	John	P4
S1	John	P5
S1	John	P6
S2	Victor	P1
S2	Victor	P2
S3	Anna	P2
S4	Angela	P3
S4	Angela	P4
S4	Angela	P5
S5	Paul	NULL

AGGREGATE QUERIES

AGGREGATE QUERIES

- **Aggregate Query**: query in which the result depends on the consideration of **sets of rows**
- The result is a single (**aggregated**) value
- Expressed in the `SELECT` clause
  - aggregate operators are evaluated on the rows accepted by the `WHERE` conditions
- SQL92 offers five aggregate operators
  - `COUNT`, `SUM`, `MAX`, `MIN`, `AVG`
- Except for `COUNT`, these functions return a `NULL` value when no rows are selected

OPERATOR COUNT

- COUNT returns the number of rows or distinct values

```
COUNT (<* | [DISTINCT | ALL] TargetList >)
```

- The DISTINCT keyword forces the count of distinct values in the attribute list

COUNT EXAMPLE /1

- Find the **number of suppliers** in the database

```
SELECT COUNT (*)
 FROM  Supplier
```

Supplier

CodeS	NameS	Shareholders	Office
S1	John	2	Amsterdam
S2	Victor	1	Den Haag
S3	Anna	3	Den Haag
S4	Angela	2	Amsterdam
S5	Paul	3	Utrecht

count
5

COUNT EXAMPLE /2

- Find the **number of suppliers** with at least one supply

```
SELECT COUNT (*)
 FROM  Supply
```

- Is it right?
- Equivalent to `SELECT COUNT(CodeP)` or `SELECT COUNT(CodeS)`

Supply

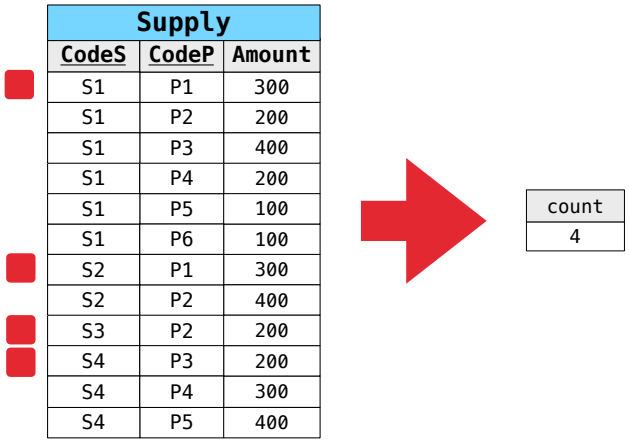
CodeS	CodeP	Amount
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

count
12

COUNT EXAMPLE /3

- Find the **number of suppliers** with at least one supply

```
SELECT COUNT (DISTINCT CodeS)
 FROM  Supply
```



Supply		
CodeS	CodeP	Amount
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

count
4

COUNT EXAMPLE /4

- **Count** the number of suppliers that supply the product "P2"

```
SELECT COUNT(*)
 FROM  Supply
 WHERE CodeP = 'P2'
```

- Is it right?

Supply

CodeS	CodeP	Amount
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

count
3

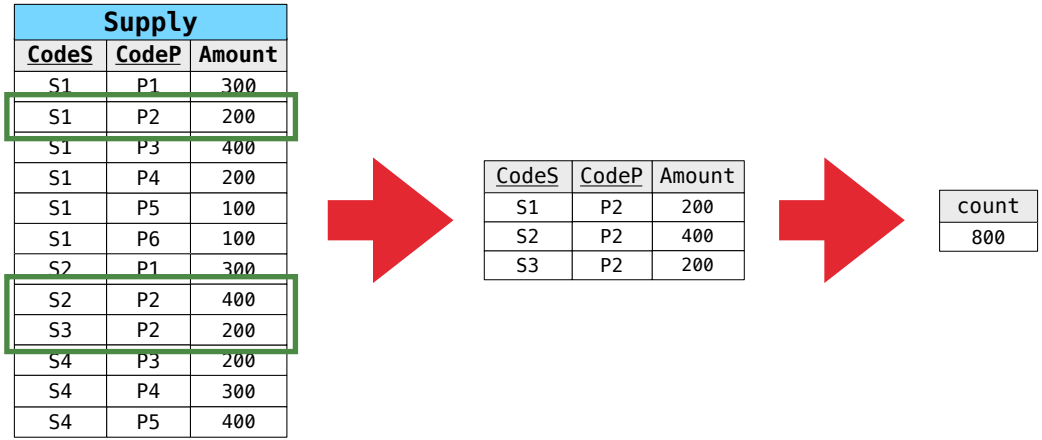
OPERATORS SUM,MAX,MIN,AVG

- `SUM,MAX,MIN,AVG`
  - Allowed arguments are attributes or expressions
- `SUM,AVG`
  - Only numeric types
- `MAX,MIN`
  - Attribute must be sortable
  - Applied also on strings and timestamps

SUM EXAMPLE

- Find the **total** number of supplied items for product "P2"

```
SELECT SUM(Amount)
 FROM  Supply
 WHERE CodeP = 'P2'
```



NULL VALUES AND AGGREGATES

- All aggregate operations ignore tuples with NULL values on the aggregated attributes
  - `COUNT`: number of input rows for which the value of expression is not `NULL`
  - `SUM`, `AVG`, `MAX`, `MIN`: `NULL` values are not considered
- The `COALESCE` function can be used to force a value for `NULL`

```
SELECT AVG(season_nr)
 FROM  title_100k
```

```
SELECT AVG(COALESCE(season_nr,1))
 FROM  title_100k
```

AGGREGATE QUERY AND TARGET LIST

- This is an incorrect query, although syntactically admissible

```
SELECT FirstName, Surname, MAX(Salary)
 FROM  Employee JOIN Department ON Dept = DeptName
 WHERE Department.City = 'London'
```

- Whose name? The target list must be homogeneous

- The GROUP BY clause will help us

GROUPING ROWS

- Queries may apply aggregate operators to subsets of rows
- **For each product** find the total amount of supplied items

```
SELECT CodeP, SUM(Amount)
 FROM  Supply
 GROUP BY CodeP
```

Supply

CodeS	CodeP	Amount
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

CodeS	CodeP	Amount
S1	P1	300
S2	P1	300
S1	P2	200
S2	P2	400
S3	P2	200
S1	P3	400
S4	P3	200
S1	P4	200
S4	P4	300
S1	P5	100
S4	P5	400
S1	P6	100

CodeP	Amount
P1	600
P2	800
P3	600
P4	500
P5	500
P6	100

GROUP BY CLAUSE /1

- The order of the grouping attributes does not matter
- The SELECT clause can contain
  - Attributes specified in the GROUP BY clause
  - Aggregated functions
  - Attributes univocally determined by attributes already specified in the GROUP BY clause

Employee

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

Department

DeptName	Address	City
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Joné

```
SELECT Office
 FROM  Employee
 GROUP BY Dept
```

- Incorrect Query

GROUP BY CLAUSE /2

- The order of the grouping attributes does not matter
- The SELECT clause can contain
  - Attributes specified in the GROUP BY clause
  - Aggregated functions
  - Attributes univocally determined by attributes already specified in the GROUP BY clause

Employee

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

Department

DeptName	Address	City
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Joné

```
SELECT DeptName,D.City,COUNT(*)
 FROM Employee E JOIN Department D ON E.Dept=D.DeptName
 GROUP BY DeptName
```

- Incorrect Query

GROUP BY CLAUSE /3

- The order of the grouping attributes does not matter
- The SELECT clause can contain
  - Attributes specified in the GROUP BY clause
  - Aggregated functions
  - Attributes univocally determined by attributes already specified in the GROUP BY clause

Employee

FirstName	Surname	Dept	Office	Salary	City
Mary	Brown	Administration	10	45	London
Charles	White	Production	20	36	Toulouse
Gus	Green	Administration	20	40	Oxford
Jackson	Neri	Distribution	16	45	Dover
Charles	Brown	Planning	14	80	London
Laurence	Chen	Planning	7	73	Worthing
Pauline	Bradshaw	Administration	75	40	Brighton
Alice	Jackson	Production	20	46	Toulouse

Department

DeptName	Address	City
Administration	Bond Street	London
Production	Rue Victor Hugo	Toulouse
Distribution	Pond Road	Brighton
Planning	Bond Street	London
Research	Sunset Street	San Joné

```
SELECT DeptName,D.City,COUNT(*)
 FROM  Employee E JOIN Department D ON E.Dept=D.DeptName
 GROUP BY DeptName, D.City
```

- Correct Query

GROUPING ROWS

- Queries may apply aggregate operators to subsets of rows
- **For each product** sold by suppliers in Den Haag, find the total amount of supplied items

```
SELECT CodeP, SUM(Amount)
 FROM  Supply JOIN Supplier ON Supply.CodeS = Supplier.CodeS
 WHERE Office = 'Den Haag'
 GROUP BY CodeP
```

Supplier				Supply		
CodeS	NameS	Shareholders	Office	CodeS	CodeP	Amount
S1	John	2	Amsterdam	S1	P1	300
S1	John	2	Amsterdam	S1	P2	200
S1	John	2	Amsterdam	S1	P3	400
S1	John	2	Amsterdam	S1	P4	200
S1	John	2	Amsterdam	S1	P5	100
S1	John	2	Amsterdam	S1	P6	100
S2	Victor	1	Den Haag	S2	P1	300
S2	Victor	1	Den Haag	S2	P1	300
S2	Victor	1	Den Haag	S2	P2	400
S3	Anna	3	Den Haag	S3	P2	200
S4	Angela	2	Amsterdam	S4	P3	200
S4	Angela	2	Amsterdam	S4	P4	300
S4	Angela	2	Amsterdam	S4	P5	400

CodeP	Amount
P1	300
P1	300
P2	200
P2	400

CodeP	Amount
P1	600
P2	600

HAVING CLAUSE /1

- Conditions on the result of an aggregate operator require the `HAVING` clause
- Only predicates containing aggregate operators should appear in the argument of the `HAVING` clause
- Find the departments in which the average salary of employees working in office number 20 **is higher than 25**

```
SELECT Dept
 FROM Employee
 WHERE Office = '20'
 GROUP BY Dept
 HAVING AVG(Salary) > 25
```

HAVING CLAUSE /2

- Find the total number of supplied items for products that count **at least 600 total supplied items**

```
SELECT CodeP, SUM(Amount)
 FROM  Supply
 GROUP BY CodeP
 HAVING SUM(Amount) >= 600
```

Supply

CodeS	CodeP	Amount
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

CodeS	CodeP	Amount
S1	P1	300
S2	P1	300
S1	P2	200
S2	P2	400
S3	P2	200
S1	P3	400
S4	P3	200
S1	P4	200
S4	P4	300
S1	P5	100
S4	P5	400
S1	P6	100

CodeP	Amount
P1	600
P2	800
P3	600

HAVING CLAUSE /3

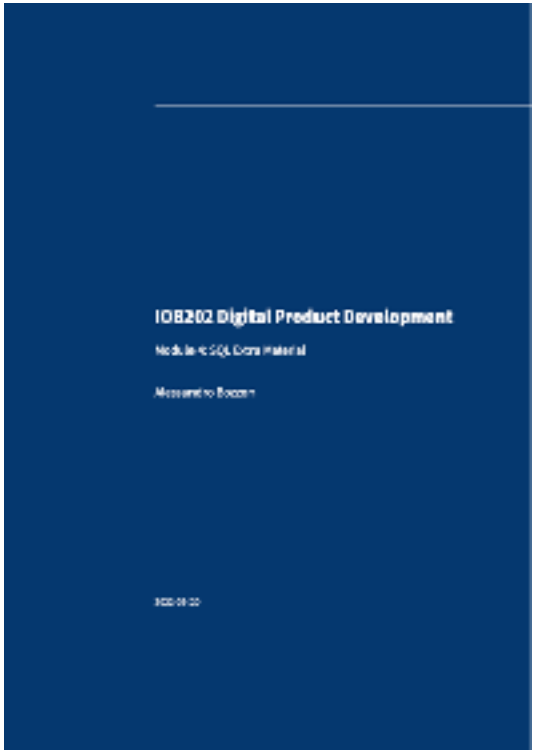
- Find the code of red products supplied by more than one supplier

```
SELECT Supply.CodeP
 FROM  Supply JOIN Products ON Supply.CodeP = Product.CodeP
 WHERE Color = 'Red'
 GROUP BY Supply.CodeP
 HAVING COUNT(*) > 1
```

Products					Supply		
CodeP	NameP	Color	Size	Storehouse	CodeS	CodeP	Amount
P1	Sweater	Red	40	Amsterdam	S1	P1	300
P1	Sweater	Red	40	Amsterdam	S2	P1	300
P2	Jeans	Green	48	Den Haag	S1	P2	200
P2	Jeans	Green	48	Den Haag	S2	P2	400
P2	Jeans	Green	48	Den Haag	S3	P2	200
P3	Shirt	Blu	48	Rotterdam	S1	P3	400
P3	Shirt	Blu	48	Rotterdam	S4	P3	200
P4	Shirt	Blu	44	Amsterdam	S1	P4	200
P4	Shirt	Blu	44	Amsterdam	S4	P4	300
P5	Skirt	Blu	40	Den Haag	S1	P5	100
P5	Skirt	Blu	40	Den Haag	S4	P5	400
P6	Coat	Red	42	Amsterdam	S1	P6	100

CodeP
P1

EXTENSIVE SQL LECTURE NOTES AVAILABLE



Resources

- "Fundamentals of Database Systems", 7th Edition, 2016 (Global Edition). Authors: *Ramez Elmasri, Shamkant Navathe*. ISBN10: 1292097612. ISBN13: 9781292097619

- “Database Systems: Concepts, Languages & Architectures". Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Riccardo Torlone

IOB2-2

Digital Product Development

TU Delft

Module 4
Data

By Alessandro Bozzon